Theoretical Computer Science 410 (2009) 5082–5092

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A sublinear-time approximation scheme for bin packing

Tuğkan Batu [a,*], Petra Berenbrink [b], Christian Sohler [c]

[a] Department of Mathematics, London School of Economics, London, UK
[b] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
[c] Department of Computer Science, TU Dortmund, Dortmund, Germany

ARTICLE INFO

Article history:
Received 30 January 2008
Received in revised form 5 June 2009
Accepted 10 August 2009
Communicated by A. Fiat

Keywords:
Sublinear-time algorithms
Bin packing

ABSTRACT

The bin packing problem is defined as follows: given a set of n items with sizes $0 < w_1, w_2, \ldots, w_n \leq 1$, find a packing of these items into a minimum number of unit-size bins possible.

We present a sublinear-time asymptotic approximation scheme for the bin packing problem; that is, for any $\epsilon > 0$, we present an algorithm A_ϵ that has sampling access to the input instance and outputs a value k such that $C_{\text{opt}} \leq k \leq (1+\epsilon) \cdot C_{\text{opt}} + 1$, where C_{opt} is the cost of an optimal solution. It is clear that uniform sampling by itself will not allow a sublinear-time algorithm in this setting; a small number of items might constitute most of the total weight and uniform samples will not hit them. In this work we use weighted samples, where item i is sampled with probability proportional to its weight: that is, with probability $w_i / \sum_i w_i$. In the presence of weighted samples, the approximation algorithm runs in $\tilde{O}(\sqrt{n} \cdot \text{poly}(1/\epsilon)) + g(1/\epsilon)$ time, where $g(x)$ is an exponential function of x. When both weighted sampling and uniform sampling are allowed, $\tilde{O}(n^{1/3} \cdot \text{poly}(1/\epsilon)) + g(1/\epsilon)$ time suffices. In addition to an approximate value to C_{opt}, our algorithm can also output a constant-size "template" of a packing that can later be used to find a near-optimal packing in linear time.

1. Introduction

The bin packing problem is a classical combinatorial optimisation problem and defined as follows: given a set X of n items with sizes $0 < w_1, w_2, \ldots, w_n \leq 1$, find a packing of these items into fewest unit-size bins possible. Bin packing has many applications in industry whenever certain items (paper, wood, pipes, etc.) can be bought only in a fixed given length and have to be cut to the lengths needed in the application. The bin packing problem is *NP*-hard and therefore, a number of approximation algorithms and heuristics have been developed: for example, first fit, best fit, sum-of-squares, or Gilmore–Gomory cuts [1,5,6,13,14].

Although these heuristics typically perform well in practice, it would be good to also have an estimate of the cost of an optimal solution in order to detect cases where the heuristic does not provide a good solution. However, we also do not want to spend much time on computing such a value. Ideally, we would like to compute such a value by taking a small sample of the input set and analyse only this sample set. In this paper we address this question of designing a sublinear-time algorithm to approximate the optimal value of a solution to the bin packing problem. We also believe that the (non-uniform) sampling process used to compute a sample set gives important insights how to obtain representative sample sets for this problem. In particular, it could be interesting to combine our sampling methods with known heuristics to approximate the cost of a solution quickly.

* Corresponding author. Tel.: +44 207 955 6540.
E-mail addresses: T.Batu@lse.ac.uk (T. Batu), petra@cs.sfu.ca (P. Berenbrink), christian.sohler@tu-dortmund.de (C. Sohler).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.08.006

Our results. We present a sublinear-time approximation scheme for the cost of the bin packing problem. We assume that the algorithm has access to an oracle that gives weighted samples of the input items; that is, the probability that the oracle returns item (i, w_i) is $w_i / \sum_j w_j$. We assume that the algorithm receives both the index and the weight of the item so that it can distinguish between different items of same weight. Our algorithm takes $\tilde{O}(\sqrt{n} \cdot \text{poly}(1/\epsilon))$ weighted samples from the input and returns an integer k such that

$$C_{\text{opt}} \leq k \leq (1+\epsilon) \cdot C_{\text{opt}} + 1,$$

where C_{opt} is the optimal value for the given problem instance.

It is clear that in the absence of an oracle that gives weighted samples as described above, a sublinear-time algorithm taking uniform samples from the input items may fail to identify the items of large size, hence, fail to output a good approximation to the number of bins needed. Note that we do not require that the algorithm knows the total weight of the items. In fact, the total weight of the items gives a 2-approximation to the optimal solution since in the optimal packing, all but one of the bins are guaranteed to be half full. We will show that one can approximate the total weight of the items using samples. Moreover, if the total weight of the items is part of the input, then our algorithm yields a constant-time approximation scheme for bin packing.

Estimating the total weight with a sample size that is linear in the reciprocal of the average weight can be done by a simple application of the Chernoff–Hoeffding bounds. However, when the total weight is small, which corresponds to an instance that fits in a small number of bins, this approach might be very inefficient. Therefore, we propose another method to estimate the total weight that uses $\tilde{O}(\sqrt{n})$ weighted samples regardless of the average weight. We then show that the uniform samples can be helpful when used together with weighted samples. We present an algorithm that has access to both uniform and weighted samples and approximates the total weight using $\tilde{O}(n^{1/3} \cdot \text{poly}(1/\epsilon))$ weighted and uniform samples. Both of these results are tight up to polylogarithmic factors in terms of n.

As it is a sublinear-time algorithm, our algorithm cannot output a description of a packing that achieves a near-optimal cost. However, it is still possible to extract a constant-size description of a "template" packing from our algorithm that can be used to find a packing (with the same cost as output) in linear time.

Related work. Our bin packing algorithm is based on the linear-time $(1+\epsilon)$-approximation algorithm of Fernandez de la Vega and Lueker [11]. The main idea behind that algorithm is to simplify the input instance by grouping all the items into a constant number of groups and using a single weight value for each item in a group. This new input instance with only a constant number of different weights is then used to obtain an approximation to the optimal value for the original input. For a survey about bin packing algorithms, see [12].

The field of sublinear algorithms studies the question of how to approximate the output for a given problem by only looking at a small random sample of the input. Sublinear algorithms are known for a number of problems from different areas. For instance, property testing algorithms are sublinear algorithms that solve relaxations of the standard decision problems. In particular, the goal of a property testing algorithm is to distinguish between the input instances that have a specific property and the input instances that are far from any input instance that has the property. Although property testing adopts this "dual" notion of approximation, the algorithms may also lead to approximations in the standard sense. For example, the property tester of Goldreich et al. [15] for ρ-cut problem yields a constant-time approximation scheme for Max-Cut problem on dense graphs. In a graph with average degree d and n vertices, it is known that one can estimate the number of connected components up to an additive error of ϵn in time polynomial in $1/\epsilon$ [16]. Based on this idea, an algorithm to estimate the weight of the minimum spanning tree has been developed, since this weight can be expressed as the sum of the number of connected components in certain subgraphs [4]. Variants of this algorithm have been designed to estimate the weight of Euclidean and metric minimum spanning trees in sublinear time [7,9]. It is also possible to estimate the average degree [17,10] of the vertices in a graph and the cost of a minimum vertex cover [24] in sublinear time.

In the area of clustering it is known that one can compute an approximate solution for the k-median and k-means problems in $\widetilde{O}(nk)$ time for the metric variant of the problem [19,21,25]. Notice that this is also sublinear in the input size because the description size of an arbitrary metric space is $\Theta(n^2)$. For the same setting, one can also approximate the cost of a basic facility location problem in $\widetilde{O}(n)$ time [2]. The quality of a uniformly distributed random sample has been analysed for a number of clustering variants [22,8]. Also, for the min-sum 2-clustering problem, a sublinear-time $(1+\epsilon)$-approximation algorithm is given [20].

In the context of property testing of distributions, Batu et al. [3] give a sublinear-time algorithm that takes samples from a discrete distribution on n items and outputs a $(1+\epsilon)$-approximation to the (Shannon) entropy of the distribution. In fact, we borrow some techniques from [3] to obtain our results in this paper.

Independently from our results, the authors of [23] present algorithms to approximate the sum of n weighted variables using weighted sampling. Their upper bounds on the sample complexity ($\tilde{O}(\sqrt{n})$ for weighted sampling and $\tilde{O}(n^{1/3})$ if both uniform and weighted samples are used) are essentially the same as our bounds. However, their algorithm is quite different from ours. The main idea of our algorithm is as follows. We divide the interval $[0, 1]$ into k sub-intervals of geometrically growing lengths, for a suitable chosen k. We assign one bucket to every interval. Then, we sample items according to their weight and sort them into the corresponding buckets. Finally, for every bucket, we use the sample frequencies to estimate the total weight contribution of items falling into the bucket. The sum of these weight contributions is used as an approximation for the total weight of all items. The main idea of the algorithm in [23] is as follows. The authors first

guess a total weight αn and fix a uniform bucket size $\epsilon\alpha$ accordingly. Every input item is then broken down into pieces of that uniform size. If the guess αn is now approximately correct, this should result in roughly ϵn uniform sized buckets. The authors use weighted sampling to sample from these uniform sized buckets, and use number of samples they need until they see a repeated bucket to decide if the initial guess is correct or not. If it is correct, they output αn, otherwise they try another guess. The sample complexity for both results are roughly the same. For the case of weighted samples only, we need $O(\sqrt{n} \cdot (\log n + \log(1/\epsilon))/\epsilon^3)$ samples, whereas the approach of [23] uses $O(\sqrt{n} \cdot \log n \cdot (\log\log n + \log(1/\epsilon))/\epsilon^{3.5})$ many samples. For the case of uniform and weighted samples, we need $O(n^{1/3} \cdot (\log n + \log(1/\epsilon))/\epsilon^3)$ samples, whereas the approach of [23] uses $O(n^{1/3} \cdot \log n \cdot (\log\log n + \log(1/\epsilon))/\epsilon^{4.5})$ many samples.

Organisation. The paper is organised as follows. In Section 2, we present our algorithm for bin packing instances where all items have large weight. In Section 3, we present our algorithm for general bin packing instances. In Section 4, we describe our algorithms for estimating the total weight of the items. Finally, in Section 5, we conclude with some implications about related problems.

2. Algorithm for packing heavy items

In the following, we consider only *h*eavy instances of bin packing problem: we assume that for all $1 \leq i \leq n$, we have $w_i \geq \gamma$. As a direct consequence, any bin can contain at most $1/\gamma$ items, which will be very useful in our analysis. We show that for heavy instances, we get a $(1+\epsilon)$-approximation algorithm with a running time *independent* of the input size. This algorithm is based on the algorithm of Fernandez de la Vega and Lueker [11].

Throughout this section, we assume that the algorithm has access to uniform samples from the items in the input; for the general algorithm in Section 3, uniform samples from the heavy items of the instance can be simulated using weighted samples (provided that the total weight of the heavy items is sufficiently high) by incurring an additional factor of $1/\gamma$ in the sample complexity. Simply, retaining to a sample of an item with weight $w \geq \gamma$ with probability γ/w ensures that we have uniform sampling from the heavy items.

The outline of the algorithm is as follows. We first use random sampling from the input to subdivide $[0, 1]$ into intervals $I_1, \ldots, I_k$ for $k = O(\epsilon^{-2})$ such that for each interval I_j, $1 \leq j \leq k$, there are roughly ϵn items whose weight lies in I_j. Let i_j be the index of the heaviest item in interval I_j. The next step is to consider a simplified input instance that has exactly ϵn items of weight w_{i_j} for each interval I_j. The optimal solution to this new instance is a good approximation to the optimal solution of our original instance. Then, we create a new "thinned-out" instance by reducing the number of items of weight w_{i_j} to some constant. Finally, we solve the problem on this new instance optimally and rescale the cost of the solution. This way we obtain a good approximation to an optimal solution for the original instance. Since we can solve the thinned-out instance in time independent of n, we have a constant-time algorithm.

2.1. Subdivision into intervals

We first describe our algorithm that subdivides $[0, 1]$ into intervals such that any interval I_j contains roughly ϵn items with weight in I_j. In the case of non-unique item weights, we use the index of the item as tie-breaker. In this case, the endpoint of an interval is given by a weight and the index of the item; that is, items with the same weight may be contained in different intervals. Our subdivision algorithm is described below.

Algorithm Subdivision(k, r)

1. Pick $s = k \cdot r$ items $i_1, \ldots, i_s$ uniformly at random with repetition from X. Let S denote the multiset of the selected items.
2. Sort items according to their weights (using indices as tie breakers).
3. Let $\langle i_{\pi(1)}, \ldots, i_{\pi(s)}\rangle$ be the resulting sequence of items.
4. Define interval $I_1 = (0, i_{\pi(r)}]$ and $I_j = (i_{\pi((j-1)\cdot r)}, i_{\pi(j\cdot r)}]$, for $2 \leq j \leq k-1$, and $I_k = (i_{\pi((k-1)\cdot r)}, 1]$.

The algorithm takes every r-th item from the sorted list S and considers the induced intervals. Note that S is a multiset since items can occur several times in S. A *boundary set* $\mathcal{B}$ is defined as a multiset of $k-1$ items from X.[1] The weights of these items can be regarded as interval boundaries since they subdivide $[0, 1]$ into k intervals $I_1, \ldots I_k$. A boundary set $\mathcal{B}$ is called *bad*, if for at least one of the intervals I_j there are either less than $\lceil(1-\epsilon\gamma)\cdot n/k\rceil$ items in X whose weight is in I_j or there are more than $\lfloor(1+\epsilon\gamma)\cdot n/k\rfloor$ items in X with a weight in I_j. Any boundary set that is not bad is called *good*, that is, it is a boundary set where every induced interval contains the weight of roughly n/k items from the input set X. Similarly, we call an interval I_j bad if it contains the weight of at most $\lceil(1-\epsilon\gamma)n/k\rceil$ or at least $\lfloor(1+\epsilon\gamma)\cdot n/k\rfloor$ items from X. The following lemma shows that with high probability, every interval I_j constructed by **Algorithm Subdivision** contains roughly n/k items from X for large enough r.

[1] For technical reasons, we allow that an item occurs more than once in $\mathcal{B}$. In this case, the corresponding interval induced by $\mathcal{B}$ is empty. If we write $\mathcal{B} \subseteq S$, we assume that the items that occur more than once in $\mathcal{B}$ must occur at least as many times in the multiset S.

Lemma 1. *Let $r = O(\frac{k\cdot\ln(k/\epsilon\gamma)}{\epsilon^2\cdot\gamma^2})$. With probability at least 7/8, we have for $1 \le j \le k$,*

$$\lceil(1-\epsilon\gamma)\cdot n/k\rceil \le \left|\left\{i : w_i \in I_j\right\}\right| \le \lfloor(1+\epsilon\gamma)\cdot n/k\rfloor.$$

Proof. We say that a boundary set $\mathcal{B}$ is *destroyed* by sample set S if at least one interval I_j induced by $\mathcal{B}$ contains either the weights of at most $(1-\epsilon\gamma/2)\cdot s/k$ items from S or the weights of more than $(1+\epsilon\gamma/2)\cdot s/k$ items from S. Our goal is to show that every bad boundary set is destroyed with high probability. A boundary set $\mathcal{B}$ is *valid* for sample set S if $\mathcal{B} \subseteq S$, *and* for each of the induced intervals I_j, S contains exactly r sample items whose weights are in I_j. Clearly, for every sample set S, there is a unique valid boundary set $\mathcal{B}_S$. For a representative sample set S, we would expect that $\mathcal{B}_S$ is good and thus, the intervals approximately subdivide the input set X into k subsets of roughly equal size.

We will consider the probability that a bad boundary set $\mathcal{B}$ is destroyed by a sample set S, conditioned on the event that $\mathcal{B} \subseteq S$. In other words, S consists of the $k-1$ interval boundaries and $s-k+1$ items chosen uniformly at random from X. Without loss of generality, we can assume that the first $k-1$ items of S are the items of $\mathcal{B}$. Let us first assume that $\mathcal{B}$ induces a bad interval I_{bad} that contains the weight of at most $(1-\epsilon)n/k$ items from X. Note that at most one of boundary items can be contained in I_{bad}. Now let us consider the remaining $s' = s-k+1$ items. Let Y_i denote the indicator random variable for the event that the weight of the ith remaining item is contained in I_{bad}. Clearly, we have

$$\Pr[Y_i = 1] \le (1-\epsilon\gamma)/k.$$

We can assume equality for the analysis. By Chernoff bounds we have

$$\Pr\left[\sum_{i=1}^{s'} Y_i \ge (1+\lambda)\cdot \mathrm{E}\left[\sum_{i=1}^{s'} Y_i\right]\right] \le e^{-\lambda^2\cdot \mathrm{E}\left[\sum_{i=1}^{s'} Y_i\right]/3}.$$

Using $\lambda = \epsilon\gamma/3$, we get

$$\Pr\left[\sum_{i=1}^{s'} Y_i \ge \left(1-\frac{\epsilon\gamma}{2}\right)s'/k\right] \le \Pr\left[\sum_{i=1}^{s'} Y_i \ge (1+\epsilon\gamma/2)\cdot(1-\epsilon\gamma)s'/k\right] \le e^{-\epsilon^2\cdot\gamma^2\cdot(1-\epsilon\gamma)\cdot s'/(27k)}.$$

Now let us assume that we have a bad interval I_{bad} induced by $\mathcal{B}$ that contains the weight of more than $(1+\epsilon)n/k$ points from X. We proceed similarly to the previous case. Again we consider the remaining $s' = s-k+1$ items and use Y_i to denote the indicator random variable for the event that the weight of the i-th remaining item is contained in I_{bad}. For this case we get

$$\Pr[Y_i = 1] \ge (1+\epsilon\gamma)/k.$$

And, again, we can assume equality for the analysis. By Chernoff bounds we have

$$\Pr\left[\sum_{i=1}^{s'} Y_i \le (1-\lambda)\cdot \mathrm{E}\left[\sum_{i=1}^{s'} Y_i\right]\right] \le e^{-\lambda^2\cdot \mathrm{E}\left[\sum_{i=1}^{s'} Y_i\right]/3}.$$

Using $\lambda = \epsilon\gamma/3$, we get

$$\Pr\left[\sum_{i=1}^{s'} Y_i \le \left(1+\frac{1}{2}\epsilon\gamma\right)s'/k\right] \le e^{-\epsilon^2\cdot\gamma^2\cdot(1-\epsilon\gamma)\cdot s'/(27k)} \le \delta.$$

Hence, we can conclude that

$$\Pr[\mathcal{B} \text{ is not destroyed} \mid \mathcal{B} \subseteq S] \le e^{-\epsilon^2\cdot\gamma^2\cdot(1-\epsilon\gamma)\cdot s'/(27k)} \le e^{-\epsilon^2\cdot\gamma^2\cdot(1-\epsilon\gamma)\cdot r/27} \le e^{-O(k\cdot\ln(k/\epsilon\gamma))} \le s^{-k}.$$

Finally, we have to show that, with a good probability, all possible bad boundary sets with $\mathcal{B} \subseteq S$ are destroyed. (Note that we assume only $\mathcal{B} \subseteq S$, we do not assume that the items of $\mathcal{B}$ are the interval borders induced by S). We call a sample S *good* if all boundary sets $\mathcal{B}$ with $k-1$ items and $\mathcal{B} \subseteq S$ destroyed. If all boundary sets $\mathcal{B} \subseteq S$ are destroyed, this also holds for the unique valid boundary set $\mathcal{B}_S$. Let $S = \{i_1, \ldots, i_s\}$.

$$\begin{aligned}
\Pr[S \text{ is good}] &\ge 1 - \sum_{\text{bad boundary set } \mathcal{B}} \Pr[\mathcal{B} \subseteq S]\cdot\Pr[\mathcal{B} \text{ is not destroyed} \mid \mathcal{B} \subseteq S] \\
&\ge 1 - n^{k-1}\cdot\binom{s}{k-1}\cdot\frac{1}{n^{k-1}}\cdot(k-1)!\cdot s^{-k} \\
&\ge 7/8,
\end{aligned}$$

where the second inequality follows from the fact that $\Pr[B \subseteq S] \le \binom{s}{k-1}\cdot\frac{1}{n^{k-1}}\cdot(k-1)!$. □

We now consider the following simplified problem instance X_{new}. For each interval $I_j = (\ell_j, r_j]$, $1 \le j \le k$, we have $\lfloor(1+\epsilon\gamma)n/k\rfloor$ items with weight *exactly* r_j. If the boundary set obtained from the sample set S is good, then this modification only increases the cost of an optimal solution. We will show that an optimal solution to our new instance X_{new} has cost at most $(1+3\epsilon)\cdot C_{\text{opt}}$, where C_{opt} denotes the cost of an optimal solution for X.

Lemma 2. *Let $k \ge 1/\epsilon\gamma$. If the boundary set used to generate X_{new} is good, then we get, for the cost C_{new} of an optimal solution for X_{new},*

$$C_{\text{opt}} \le C_{\text{new}} \le (1+3\epsilon)\cdot C_{\text{opt}}.$$

Proof. Since the boundary set used is good, we know that every interval I_j contains the weight of at least $\lceil(1-\epsilon\gamma)\cdot n/k\rceil$ items. Let us consider an optimal assignment of items from X to bins. We use this assignment to pack most of the items from X_{new}. For each r_j, $1 \le j \le k-1$, we select $\lceil(1-\epsilon\gamma)\cdot n/k\rceil$ items with weight r_j and pack these items into the bins that are used for items with weight in I_{j+1} in an optimal solution for X. Clearly, this way we use at most as many bins as used in an optimal solution for X. At this point we have packed all but $2\epsilon\gamma n/k$ items whose weight is in the first $k-1$ intervals. Thus, $2(k-1)\epsilon\gamma n/k \le 2\epsilon\gamma n$ plus the n/k items with weight in I_k remain. Each of these items is placed in an individual bin. Since $C_{\text{opt}} \ge \gamma n$ and $k \ge 1/(\epsilon\gamma)$, we get that these items incur a cost of at most $3\epsilon\cdot C_{\text{opt}}$. □

It remains to show that we can use scaling to approximate the cost of an optimal solution. Let $F(\gamma, k)$ denote the set of combinatorial distinct fillings of a bin with at most $1/\gamma$ items when we have k different types of items and let $f(\gamma, k) = |F(\gamma, k)| = O(k^{1/\gamma})$. We prove the following lemma.

Lemma 3. *Let ℓ be integer. Given an instance X_1 of the bin packing problem that consists of $m = \lceil\frac{f(\gamma,k)}{\epsilon\cdot\gamma\cdot k}\rceil\cdot\ell$ copies of each of k items with weights $w_1, \ldots, w_k \in [\gamma, 1]$. Furthermore, let X_2 be an instance that consists of $\lceil\frac{f(\gamma,k)}{\epsilon\cdot\gamma\cdot k}\rceil$ copies of the same set of items. Let $C_{\text{opt}}^{(X_1)}$ and $C_{\text{opt}}^{(X_2)}$ denote the costs of the optimal solution of X_1 and X_2, respectively. Then,*

$$C_{\text{opt}}^{(X_1)} \le \ell\cdot C_{\text{opt}}^{(X_2)} \le (1+\epsilon)\cdot C_{\text{opt}}^{(X_1)}.$$

Proof. Let us consider an optimal solution for X_2. We copy this solution ℓ times. This will give us a solution for instance X_1 with cost $\ell\cdot C_{\text{opt}}^{(X_2)}$. This shows the first inequality.

To show the second inequality, let us consider an optimal solution for X_1. For each filling type $f_i \in F(\gamma, k)$, we round up the number of bins that have that filling type in our optimal solution to the nearest multiple of ℓ. Let j_i be that number. To obtain a solution for X_2, we use j_i/ℓ many bins of type f_i. Clearly, the cost of the rounded solution is at most an additive term of $\ell\cdot f(\gamma, k)$ larger than the optimal solution for X_1. Since we have $m = k\cdot\lceil f(\gamma, k)/(\epsilon\cdot\gamma\cdot k)\rceil\cdot\ell$ items, we have $\epsilon\cdot C_{\text{opt}}^{(X_1)} \ge \epsilon\cdot\gamma\cdot m \ge \ell\cdot f(\gamma, k)$, which proves the second inequality. □

Now we are ready to describe our algorithm for heavy items.

Algorithm Heavy Items

1. Use **Subdivision**(k, r) with $k = 1/\epsilon\gamma$ and $r = O(k\ln(1/\epsilon\gamma)/\epsilon^2\gamma^2)$ to obtain intervals I_j, $1 \le j \le k$.
2. Define X' to be an instance containing $\lfloor(1+\epsilon\gamma)n/k\rfloor$ items with weight r_j for each interval $I_j = (\ell_j, r_j]$, $1 \le j \le k$.
3. Create a new instance X'' that has $\lceil\frac{f(\gamma,k)}{\epsilon\cdot\gamma\cdot k}\rceil$ copies of every item with weight r_j.
4. Compute the optimal number d of bins for X'' and output $d\cdot\lceil\frac{(1+\epsilon\gamma)\cdot n\cdot\epsilon\cdot\gamma}{f(\gamma,k)}\rceil$.

Theorem 4. *Let $1 > \gamma > 0$ be a constant. Given oracle access to n items with weights from $[\gamma, 1]$,* **Algorithm Heavy Items** *outputs in time $g'(\varepsilon, \gamma)$ a value b such that, with high probability,*

$$C_{\text{opt}} \le b \le (1+\varepsilon)\cdot C_{\text{opt}},$$

where C_{opt} is the cost of an optimal bin packing of the given n items and g' is a function depending on ε and γ but is independent of n.

Proof. In our first step (the subdivision algorithm) and the reduction to problem instance X_{new}, we increase the cost of the solution by at most a factor of $(1+3\epsilon)$ by Lemma 2. To apply Lemma 3, we must have that the number of occurrences of each item is a multiple of $\lceil\frac{f(\gamma,k)}{\epsilon\cdot\gamma\cdot k}\rceil$, which costs us another $(1+\epsilon)$ factor, provided that the number of input items is large enough, that is, $n \ge \lceil\frac{f(\gamma,k)}{\epsilon\cdot\gamma\cdot k}\rceil\cdot\frac{k}{\epsilon}$ (otherwise, we can solve the problem brute force). Finally, by Lemma 3, we lose another $(1+\epsilon)$ factor by scaling down the input instance. Overall, the optimal solution to our new instance may be at most $(1+\epsilon)^2\cdot(1+3\epsilon) \le (1+27\epsilon)$ larger than the optimal solution of our input instance. Using $\epsilon = \varepsilon/27$, the proof follows. □

The instance X'' has $O(f(\gamma, k)/\epsilon\gamma)$ items in it. Hence, the optimal solution to X'' can be computed via dynamic programming in time $g'(\epsilon, \gamma) = O((f(\gamma, k))^{k+1}) = 2^{O(\log(1/\epsilon\gamma)/\epsilon\gamma^2)}$ using techniques from [18].

3. General bin packing algorithm

In this section we present our algorithm for the general bin packing problem. We first observe that, if all the items have small weight, say less than some small constant γ, then it is easy to pack all these items such that each bin is filled up to $1-\gamma$ level. Hence, the total weight of the items can be used to obtain a $(1-\gamma)^{-1}$-approximation. On the other hand, if all the items have weight larger than some threshold γ (heavy item), we have seen in Section 2 that we can obtain a $(1+O(\gamma))$-approximation.

Our algorithm first tries to identify whether the input falls into one of the two cases above. Namely, if all but a small fraction of the items are light or heavy, we can ignore this minority. We can obtain a good approximation by considering only the dominant-type items. When both light and heavy items constitute significant fractions of the input, then they both have to be considered. In this case, we utilise the fact that both the total weight and the optimal packing of heavy items are lower bounds for the cost of an optimal solution. We will use the algorithms presented in Section 4 to obtain an estimate for the total weight of the items.

Let ϵ be a constant and $\gamma = \epsilon/c$ for some $c > 1$. Call an item i *light* if $w_i \leq \gamma$, call it *heavy* otherwise. Also, define $W = \sum_i w_i$, $W_h = \sum_{\text{heavy } i} w_i$, and $\alpha_h = W_h/W$.

Bin Packing Algorithm

1. Take $t = O(1/\gamma)$ weighted samples and let s be the number of heavy samples. Use $\tilde{\alpha}_h = s/t$ as an approximation of α_h.
2. Obtain estimate $\tilde{W}$ for the total weight W (as in Section 4).
3. If $\tilde{\alpha}_h < \gamma$, let $b = 0$. Otherwise, pack the heavy items using **Algorithm Heavy Items** of Section 2 within an approximation factor of $1+\gamma$. Let b be the number of required bins calculated by that algorithm.
4. Output $k = \lceil (1+2\gamma) \cdot \max\{\tilde{W}, b\} \rceil$.

Theorem 5. *For every $\epsilon \leq 1$, given oracle access to n items with weights from $(0, 1]$, the algorithm above computes a value k such that, with probability at least 3/4,*

$$C_{\text{opt}} \leq k \leq (1+\epsilon) \cdot C_{\text{opt}} + 1,$$

where C_{opt} is the cost of an optimal bin packing of the n items. The running time of the algorithm is $g(\epsilon) + \tilde{O}(\sqrt{n}/\epsilon^5)$ when only weighted samples are used and $g(\epsilon) + \tilde{O}(n^{1/3}/\epsilon^5)$ when both weighted and uniform samples are used, where $g(\epsilon)$ is an exponential function of $1/\epsilon$.

Proof. We use $\gamma = \epsilon/4$. The first lemma shows that with a probability of $11/12$, t weighted samples are sufficient to calculate a good estimation for α_h.

Lemma 6. *Assume $\alpha_h \geq \gamma/2$. Then, with probability of $11/12$, Step 1 of the algorithm calculates $\tilde{\alpha}_h$ such that*

$$\frac{3}{4}\alpha_h \leq \tilde{\alpha}_h \leq \frac{5}{4}\alpha_h.$$

Furthermore, when $\alpha_h < \gamma/2$, $\Pr[\tilde{\alpha}_h \geq (3/4) \cdot \gamma] \leq 11/12$.

Proof. We define t random variables $X_1 \ldots X_t$ with $X_i = 1$ if the weight of the ith sample is more than γ and zero otherwise. $X = \sum_{i=1}^{t} X_i$ and $E[X] = t \cdot \alpha_h$. Using Chernoff bounds, we get

$$\Pr[|X - E[X]| \geq E[X]/4] = \Pr[|X - t \cdot \alpha_h| \geq (t/4) \cdot \alpha_h] \leq 2 \cdot e^{(1/4)^2 \cdot t \cdot \alpha_h/3}.$$

For $\alpha_h \geq \gamma/2$, this probability is at most $11/12$ with $t = O(1/\gamma)$. The rest of the lemma follows since $\Pr[\tilde{\alpha}_h \geq (3/4) \cdot \gamma]$ for $\alpha_h < \gamma/2$ is not larger that the probability for the same event with $\alpha_h = \gamma/2$. □

Next, notice that the **Algorithm Heavy Items** of Section 2 is called only when $\alpha_h > 3 \cdot \gamma/4$ by Lemma 6. Also, notice that **Algorithm Heavy Items** uses uniform samples from the heavy items. It is not hard to see that uniform samples from heavy items can be obtained from weighted samples by paying an additional multiplicative factor $1/\gamma^2$ (one $1/\gamma$ for getting enough heavy samples and another $1/\gamma$ for filtering) in the sample complexity. Therefore, the use of **Algorithm Heavy Items** with only weighted samples is still valid.

Now, we show that the output k of the algorithm satisfies $C_{\text{opt}} \leq k \leq (1+\epsilon) \cdot C_{\text{opt}} + 1$. Let C_h be the cost of an optimal bin packing when the input instance is restricted to the heavy items.

We know that $\lceil W \rceil \leq C_{\text{opt}}$, $C_h \leq C_{\text{opt}}$, $W \leq \tilde{W} \leq (1+\gamma) \cdot W$, and $b \leq (1+\gamma) \cdot C_h$. Hence,

$$\begin{aligned} k &= \lceil (1+2\gamma) \cdot \max\{\tilde{W}, b\} \rceil \\ &\leq \lceil (1+2\gamma) \cdot (1+\gamma) \cdot \max\{W, C_h\} \rceil \\ &\leq (1+4\gamma) \cdot \lceil \max\{W, C_h\} \rceil + 1 \\ &\leq (1+\epsilon) \cdot C_{\text{opt}} + 1. \end{aligned}$$

The second to last inequality above follows because for $\alpha, x > 0$, $\lceil \alpha \cdot x \rceil \leq \alpha \lceil x \rceil + 1$.

Now suppose $C_{\text{opt}} = C_h$; that is, the cost of an optimal solution to only heavy items is as large as the cost of an optimal solution to all the items. Then, by Theorem 4 the fact that $C_h \leq b$,

$$C_{\text{opt}} = C_h \leq \lceil (1+2\gamma) \cdot \max\{\tilde{W}, b\} \rceil = k.$$

If $C_h < C_{\text{opt}}$, then we can construct a packing such that all but one of the bins is filled up to at least $1-\gamma$ level (by starting from C_h bins partially filled with heavy items and opening new bins as necessary). Therefore, in this case,

$$C_{\text{opt}} \leq \left\lceil \frac{W}{1-\gamma} \right\rceil \leq \lceil (1+2\gamma) \cdot \max\{\tilde{W}, b\} \rceil = k.$$

The dependence of the running time of the algorithm on n is determined by the number of samples used to estimate W. The dependence of the running time on ϵ is exponential and arises from **Algorithm Heavy Items**; namely, $g(\epsilon) = 2^{O(\log(1/\epsilon)/\epsilon^3)}$. □

Since the lower-bound arguments for total weight estimation in Section 4 directly translates to lower bounds for sample complexity of bin packing, the running time of our algorithm is tight up to polylogarithmic factors in terms of n for both kinds of sampling access.

Remark 1. The solution obtained for the scaled-down bin packing instance can be used as a constant-size "template" of a packing. The grouping of the items, scaling-down factor, and this solution to the scaled-down instance can be used in conjunction to pack all the items in linear time while achieving the cost that was output by the algorithm.

Remark 2. Note that estimating the total weight of the items dominates the time and the sample complexity of our algorithm. If the total weight is given as part of the input, we have an approximation scheme for bin packing with running time independent of the input size.

4. Estimating the total weight of the items

At first we give a simple lower bound on the number of samples needed to estimate the total weight of n items.

Observation 1. *To estimate the total weight of n items up to a constant factor, at least $\Omega(\sqrt{n})$ weighted samples or a linear number of uniform samples are required.*

Proof. Consider the bin packing instance of k items of the same weight w and $n-k$ items of weight almost 0. It is clear that estimating the total weight of these items with weighted samples amounts to approximately counting the number of items with weight w. By the birthday problem, such a counting will require $\Omega(\sqrt{k})$ samples. The first result now follows with $k = O(n)$. For uniform samples, we set k to be a constant. Then, we need a linear number of samples to draw the first item with weight w. □

The following lemma from Batu et al. [3] presents an algorithm that given uniform samples from a set of items, approximately counts the number of items.

Lemma 7 ([3]). *For every $\epsilon > 0$, there exists an algorithm that, given access to uniform samples from k items (where k is unknown to the algorithm), outputs ℓ such that $k \leq \ell \leq (1+\epsilon) \cdot k$ with probability at least $1-\delta$ using $O((\sqrt{k}/\epsilon) \cdot \log(1/\delta))$ samples.*

Next, we describe our algorithm that approximates the total weight of the items within a factor of $(1+\epsilon)$ using $\tilde{O}(\sqrt{n} \cdot \text{poly}(1/\epsilon))$ weighted samples. The main idea behind the algorithm is to partition the items into buckets according to their weights such that the weights of the items in a bucket are of the same magnitude; in particular, the ratio between the weights of any two items in the same bucket is bounded. Any bucket with a significant total weight will be well represented in the sample set. Then, we filter the samples from every bucket, so that we get a uniform distribution over all items from the bucket. These uniform samples are then passed to the algorithm from Lemma 7 to approximately count the items in each bucket. Finally, we can combine all these counts to estimate the total weight.

Let $\beta = \epsilon/c$ for some constant $c > 1$. The number of items $|B|$ in B is called the *size* of B, and the sum of the weights of items in B is called *weight* of B in the following. We define $B_0 = \{i : w_i \leq \beta/n\}$. For $t = \lceil \log_{(1+\beta)}(n/\beta) \rceil$ and $j = 1, \ldots, t$,

$$B_j = \left\{ i : w_i \in \left(\frac{\beta \cdot (1+\beta)^{j-1}}{n}, \frac{\beta \cdot (1+\beta)^{j}}{n} \right] \right\}.$$

Algorithm Total Weight Approximation — Weighted Samples

1. Take $m = \Theta(\sqrt{n} \cdot (\log n + \log(1/\epsilon))/\epsilon^3)$ independent weighted samples from the items, and let S denote the multiset of the selected items.
2. For $j = 1, \ldots, t = \lceil \log_{(1+\beta)}(n/\beta) \rceil$, let $S_j = S \cap B_j$.
3. Let $H = \{j : |S_j| > \frac{\beta}{2} \cdot \frac{m}{t}\}$.
4. For each $j \in H$ and $i \in S_j$, remove i from S_j with probability $1 - \frac{\beta \cdot (1+\beta)^{j-1}}{n \cdot w_i}$. Call the resulting multiset S'_j.
5. Run the algorithm of Lemma 7 on each S'_j to get an estimate k_j of $|B_j|$ for $j \in H$.
6. Output $\frac{1}{1-2\beta} \sum_{j \in H} \frac{\beta \cdot (1+\beta)^j \cdot k_j}{n}$.

The sample and runtime complexity of the algorithm is $O(\sqrt{n} \cdot (\log n + \log(1/\epsilon))/\epsilon^3)$. The next lemma proves the correctness of the algorithm above.

Lemma 8. *For every $\epsilon \leq 1$ and $\beta \leq \epsilon/6$, the above algorithm outputs a value $\widetilde{W}$ such that, with probability at least 3/4, $\sum_i w_i \leq \widetilde{W} \leq (1+\epsilon)\sum_i w_i$, given that $\sum_i w_i \geq 1$.*

Proof. Fix $j \in H$. We first show that the items of S'_j are chosen uniformly and independently at random from B_j. Fix $i \in B_j$. The probability that we choose item i at any step during the initial sampling is $w_i / \sum_i w_i$. After filtering, item i will be transferred to S'_j with probability $\beta \cdot (1+\beta)^{j-1}/(n \cdot w_i)$. Hence, the probability that item i is in S'_j is

$$\frac{w_i}{\sum_i w_i} \cdot \frac{\beta \cdot (1+\beta)^{j-1}}{n \cdot w_i} = \frac{\beta \cdot (1+\beta)^{j-1}}{n \cdot \sum_i w_i},$$

which is identical for any $i' \in B_j$. Hence, S'_j is a set of uniform and independent samples from B_j.

Next, we show that $|S'_j| = \Omega(\sqrt{n}/\beta)$ so that the Step (5) of the algorithm can be performed with high enough confidence. Since we have that (i) $|S_j| > \beta \cdot m/(2 \cdot t)$; (ii) the probability that a particular occurrence of an item is removed from S_j is at most $1 - (1+\beta)^{-1}$; and (iii) $t = O(\log(n/\beta)/\beta)$, we have that

$$\mathrm{E}[|S'_j|] \geq \frac{\beta \cdot m}{2 \cdot t} \cdot \frac{1}{1+\beta} = \Omega(\sqrt{n}/\beta).$$

Hence, using Chernoff bounds, we can show that $|S'_j| = \Omega(\sqrt{n}/\beta)$ with probability at least $1 - 1/12t$, and by union bound, $|S'_j| = \Omega(\sqrt{n}/\beta)$ for all $j \in H$ with probability at least $11/12$. Given that $|S'_j| = \Omega(\sqrt{n}/\beta)$, the algorithm from Lemma 7 will return k_j such that

$$|B_j| \leq k_j \leq (1+\beta) \cdot |B_j|, \tag{1}$$

for all $j \in H$ with probability at least $11/12$.

Note that $\sum_{i \in B_0} w_i \leq \beta$. For $j \notin H$, $|S_j| \leq (\beta \cdot m)/(2 \cdot t)$. Hence, by Chernoff bounds, we can show that with probability at least $1 - 1/12t$,

$$\sum_{i \in B_j} w_i \leq \frac{\beta}{t} \cdot \sum_i w_i.$$

Therefore, with probability at least $11/12$,

$$\sum_{j \notin H} \sum_{i \in B_j} w_i \leq \beta + \beta \cdot \sum_i w_i, \tag{2}$$

where the leading term β appears as an upper bound on the weight of B_0.

Now, assuming Eqs. (1) and (2) hold, we are ready to show that the output of the algorithm is a good approximation to the total weight. Since

$$\sum_i w_i = \sum_{j \in H} \sum_{i \in B_j} w_i + \sum_{j \notin H} \sum_{i \in B_j} w_i \leq \sum_{j \in H} \sum_{i \in B_j} w_i + (\beta + \beta \sum_i w_i)$$

and $\sum_i w_i \geq 1$, it follows that

$$\sum_{j \in H} \sum_{i \in B_j} w_i \geq (1 - 2\beta) \cdot \sum_i w_i. \tag{3}$$

Using Eq. (1), we can show that

$$\begin{aligned} \sum_{j \in H} \sum_{i \in B_j} w_i &\leq \sum_{j \in H} \sum_{i \in B_j} \frac{\beta \cdot (1+\beta)^j}{n} \leq \sum_{j \in H} \frac{\beta \cdot (1+\beta)^j \cdot |B_j|}{n} \\ &\leq \sum_{j \in H} \frac{\beta \cdot (1+\beta)^j \cdot k_j}{n}. \end{aligned}$$

Combining this with Eq. (3), we see that the output of the algorithm is at least $\sum_i w_i$. In the other direction, again using Eq. (1), we have that

$$\begin{aligned}\frac{1}{1-2\beta}\sum_{j\in H}\frac{\beta\cdot(1+\beta)^j\cdot k_j}{n} &\le \frac{(1+\beta)^2}{1-2\beta}\sum_{j\in H}\frac{\beta\cdot(1+\beta)^{j-1}\cdot|B_j|}{n}\\ &\le \frac{(1+\beta)^2}{1-2\beta}\sum_{j\in H}\sum_{i\in B_j} w_i \le (1+6\beta)\cdot\sum_i w_i\\ &\le (1+\epsilon)\cdot\sum_i w_i,\end{aligned}$$

where the last two inequalities follow from $\beta \le \epsilon/6 \le 1/6$. The probability that either Eq. (1) or Eq. (2) does not hold is at most 1/4. Hence, the lemma follows. □

Using Observation 1, we see that the algorithm above has an optimal sample complexity up to polylogarithmic factors in terms of the dependence on n. The algorithm is given access only to weighted samples from the input items. One immediate question we can ask is, what if the algorithm had access to both uniform and weighted samples. The $\Omega(\sqrt{n})$ lower bound for weighted-sampling algorithm arises from distinguishing an instance with n items of weight 1 from an instance of $n/2$ items with weight 1 and $n/2$ items of weight close to 0. The uniform sampling would easily distinguish these two instances. The following observation states a lower bound for algorithms that are allowed to use both weighted and uniform sampling.

Observation 2. *If the algorithm has access to both uniform and weighted samples, $\Omega(n^{1/3})$ samples are required to estimate the total weight of n items up to a constant factor.*

Proof. Consider the following two instances of the problem: (1) $n^{2/3}$ items of weight 1, $n - n^{2/3}$ items of weight almost 0; and (2) $2n^{2/3}$ items of weight 1, $n - 2n^{2/3}$ items of weight almost 0. After taking only $o(n^{1/3})$ samples from either instance, with some constant probability, none of the uniform samples hits an item of weight 1, and none of the samples (weighted or uniform) hits an item that was sampled before (this holds by the Birthday Problem). Hence, the samples from both cases are identically distributed, and the instances are indistinguishable with only $o(n^{1/3})$ samples. □

Next, we show how to extend the algorithm above to construct an algorithm that has access to both weighted and uniform samples and has a smaller sample complexity. Note that uniform sampling can be used to estimate the bucket sizes, as long as the bucket size is large enough; that is, $O(n/k)$ uniform samples are enough to obtain a reliable estimate for a bucket with at least k items. We also know that $O(\sqrt{k})$ weighted samples yield a good estimate as long as the weight of the bucket is large enough. In the following, we will construct an algorithm that uses the tradeoff between these two approaches. We will use the counting algorithm (from Lemma 7) only when we have to count $O(n^{2/3})$ items—thus, we will need $O(n^{1/3})$ weighted samples. Otherwise, items in a bucket with more than $\Omega(n^{2/3})$ items will be counted using uniform samples.

Algorithm Total Weight Approximation — Uniform and Weighted Samples

1. Take $m_1 = \Theta(n^{1/3}\cdot(\log\log n + \log(1/\epsilon))/\epsilon^2)$ independent uniform samples from the items, and let R denote the multiset of the selected items.
2. For $j = 1, \ldots, t = \lceil \log_{(1+\beta)}(n/\beta)\rceil$, let $R_j = R \cap B_j$. Let $L = \{j : |R_j| > \frac{1}{4}\cdot m_1 \cdot n^{-1/3}\}$.
3. For $j \in L$, let $k_j = (1+\beta/2)\cdot|R_j|\cdot n/m_1$.
4. Take $m_2 = \Theta(n^{1/3}\cdot(\log n + \log(1/\epsilon))/\epsilon^3)$ independent weighted samples from the items, and let S denote the multiset of the selected items.
5. For $j = 1, \ldots, t = \lceil \log_{(1+\beta)}(n/\beta)\rceil$, let $S_j = S \cap B_j$. Let $H = \{j : |S_j| > \frac{\beta}{2}\cdot\frac{m_2}{t}\} \setminus L$.
6. For each $j \in H$ and $i \in S_j$, remove i from S_j with probability $1 - \frac{\beta\cdot(1+\beta)^{j-1}}{n\cdot w_i}$. Call the resulting multiset S'_j.
7. Run the algorithm of Lemma 7 on each S'_j to get an estimate k_j of $|B_j|$ for $j \in H$.
8. Output $\frac{1}{1-2\beta}\sum_{j\in L\cup H}\frac{\beta\cdot(1+\beta)^j\cdot k_j}{n}$.

The sample and runtime complexity of the algorithm is $O(n^{1/3}\cdot(\log n + \log(1/\epsilon))/\epsilon^3)$. The next lemma proves the correctness of the algorithm.

Lemma 9. *For every $\epsilon \le 1$ and $\beta \le \epsilon/6$, the above algorithm outputs a value $\widetilde{W}$ such that, with probability at least 3/4, $\sum_i w_i \le \widetilde{W} \le (1+\epsilon)\sum_i w_i$, given that $\sum_i w_i \ge 1$.*

Proof. Fix j and suppose $|B_j| \ge n^{2/3}/4$. Then, $\mathrm{E}[|R_j|] = (|B_j|/n)\cdot m_1 \ge m_1\cdot n^{-1/3}/4$. By Chernoff bounds and $\log t = O(\log\log n + \log(1/\beta))$, we have that with probability at least $1 - 1/16t$,

$$(1-\beta/4)\cdot|B_j| \le |R_j|\cdot\frac{n}{m_1} \le (1+\beta/4)\cdot|B_j|.$$

Therefore, we have $j \in L$ for any j with $|B_j| \geq n^{2/3}/2$, and for any $j \in L$ we have $|B_j| \geq n^{2/3}/5$. Finally, we conclude that with probability at least 15/16, for all $j \in L$,

$$|B_j| \leq k_j \leq (1+\beta) \cdot |B_j|. \tag{4}$$

The rest of the proof is analogous to the proof of Lemma 8. Using the fact that for $j \in H$, $|B_j| \leq n^{2/3}$, we can show that, with probability of at least 14/16, m_2 weighted samples are enough to guarantee that

$$|B_j| \leq k_j \leq (1+\beta) \cdot |B_j|, \tag{5}$$

for all $j \in H$. Similarly, with probability at least 15/16,

$$\sum_{j \notin L \cup H} \sum_{i \in B_j} w_i \leq \beta \cdot \sum_i w_i. \tag{6}$$

Given that Eqs. (4), (5), and (6) hold, it follows that

$$\sum_{j \in L \cup H} \sum_{i \in B_j} w_i \geq (1-2\beta) \cdot \sum_i w_i. \tag{7}$$

Once again, analogous to the earlier proof, we can conclude that

$$\sum_{j \in L \cup H} \sum_{i \in B_j} w_i \leq \sum_{j \in L \cup H} \frac{\beta \cdot (1+\beta)^j \cdot k_j}{n}$$

and

$$\frac{1}{1-2\beta} \sum_{j \in L \cup H} \frac{\beta \cdot (1+\beta)^j \cdot k_j}{n} \leq (1+\epsilon) \sum_i w_i.$$

The error probability is bounded by 1/4. □

5. Conclusions

A natural question to ask is what other combinatorial optimisation problems yield themselves to sublinear-time algorithms? And, perhaps, what modes of access to the input is required to solve each problem, if at all?

One problem to consider is the *vector packing problem*, a generalisation of the bin packing. The input to a vector packing problem is a set of n vectors of d dimensions with values in $(0, 1]$. Each dimension describes a bin packing problem that has to be solved with all the other dimensions in parallel; that is, find an assignment of vectors to a minimum number of bins such that in each bin and for each dimension, the sum of vector entries along that dimension is at most 1. Given that we can sample items (i.e., vectors) weighted proportionally to the their L_∞ norms, we can get a $d(1+\epsilon)$-approximation to the vector packing problem. The details of this result again follows [11] and is relatively straightforward.

Another problem with similar flavour is the minimum makespan scheduling problem, defined as follows: given processing times $p_1, p_2, \ldots, p_n$ for n jobs and an integer m, find an assignment of the jobs to m identical machines so that the completion time (makespan) is minimised. The close connection between the makespan scheduling and bin packing problem is clear. Namely, the makespan scheduling problem can be viewed as finding the minimum t such that the n input jobs can be packed in m bins of size t each.

For a given makespan scheduling instance, two critical quantities are the average load $\bar{L}$ of a machine; that is, $\bar{L} \stackrel{\text{def}}{=} \sum_i p_i/m$, and the maximum processing time $q \stackrel{\text{def}}{=} \max_i p_i$. It is clear that both q and $\bar{L}$ are lower bounds for the minimum makespan. Moreover, $q + \bar{L}$ is an upper bound on the minimum makespan and it also is a 2-approximation. We can easily adjust our weighted samples algorithm to determine whether either q or $\bar{L}$ dominates the other, and then output an approximation to the cost of an optimal solution if that is the case. Otherwise, we can make the following observation: any job i such that $p_i > \beta \cdot \bar{L}$ for some parameter β is observed in a large enough sample set of size independent of n, with high probability. We could now try to estimate the makespan resulting from all smalls jobs with $p_i \leq \beta \cdot \bar{L}$, and then to estimate the time it takes to schedule remaining jobs all of which have been observed in our sample and have a processing time p_i with $\beta \cdot \bar{L} \leq p_i \leq \beta^{-1} \cdot \bar{L}$. This latter problem is seemingly hard to solve even in linear time in m.

Acknowledgements

We thank Gábor Tardos for simplifying our general bin packing algorithm.
The third author's research was supported by DFG grant SO 514/3-1 'Sublinear-time algorithms'.

References

[1] D. Applegate, L. Buriol, B. Dillard, D. Johnson, P. Shor, The cutting-stock approach to bin packing: Theory and experiments, in: Proceedings of Algorithm Engineering and Experimentation (ALENEX), 2003, pp. 1–15.
[2] M. Bădoiu, A. Czumaj, P. Indyk, C. Sohler, Facility location in sublinear time, in: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, 2005, pp. 866–877.
[3] Tuğkan Batu, Sanjoy Dasgupta, Ravi Kumar, Ronitt Rubinfeld, The complexity of approximating the entropy, SIAM Journal on Computing 35 (1) (2005) 132–150.
[4] B. Chazelle, R. Rubinfeld, L. Trevisan, Approximating the minimum spanning tree weight in sublinear time, in: Proceedings of the 28th International Colloquium on Automata, Languages and Programming, 2001, pp. 190–200.
[5] J. Csirik, D. Johnson, C. Kenyon, J. Orlin, P. Shor, R. Weber, On the sum-of-squares algorithm for bin-packing, in: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC), 2000, pp. 208–217.
[6] J. Csirik, D. Johnson, C. Kenyon, P. Shor, R. Weber, A self-organizing bin packing heuristic, in: Proceedings of Algorithm Engineering and Experimentation (ALENEX), 1999.
[7] A. Czumaj, F. Ergun, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, C. Sohler, Approximating the weight of the euclidean minimum spanning tree in sublinear time, SIAM Journal on Computing 34 (1) (2005) 91–109.
[8] A. Czumaj, C. Sohler, Sublinear-time approximation algorithms for clustering via random sampling, Random Structures & Algorithms 30 (1–2) (2007) 226–256.
[9] A. Czumaj, C. Sohler, Estimating the weight of metric minimum spanning trees in sublinear-time, in: Proceedings of the 36th ACM Symposium on Theory of Computing, 2004, pp. 175–183.
[10] U. Feige, On sums of independent random variables with unbounded variance, and estimating the average degree in a graph, in: Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 594–603.
[11] W. Fernandez de la Vega, G.S. Lueker, Bin packing can be solved within 1+epsilon in linear time, Combinatorica 1 (4) (1981) 349–355.
[12] M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing problems: A survey, in: G. Ausiello, M. Lucertini (Eds.), Analysis and Design of Algorithms in Combinatorial Optimization, Springer-Verlag, 1981, pp. 147–172.
[13] P. Gilmore, R. Gomory, A linear programming approach to the cutting-stock problem, Operations Research 9 (1961) 849–859.
[14] P. Gilmore, R. Gomory, A linear programming approach to the cutting-stock problem – Part II, Operations Research 11 (1963) 863–888.
[15] O. Goldreich, S. Goldwasser, D. Ron, Property testing and its connection to learning and approximation, Journal of the ACM 45 (4) (1998) 653–750.
[16] O. Goldreich, D. Ron, Property testing in bounded degree graphs, Algorithmica 20 (2) (2002) 165–183.
[17] O. Goldreich, D. Ron, Approximating average parameters of graphs, in: Proceedings of 10th RANDOM, 2006, pp. 363–374.
[18] D.S. Hochbaum, D.B. Shmoys, Using dual approximation algorithms for scheduling problems theoretical and practical results, Journal of the ACM 34 (1) (1987) 144–162.
[19] P. Indyk, Sublinear time algorithms for metric space problems, in: Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), 1999, pp. 428–434.
[20] P. Indyk, A sublinear time approximation scheme for clustering in metric spaces, in: Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS), 1999, pp. 154–159.
[21] R.R. Mettu, C.G. Plaxton, Optimal time bounds for approximate clustering, Machine Learning 56 (1–3) (2004) 35–60.
[22] N. Mishra, D. Oblinger, L. Pitt, Sublinear time approximate clustering, in: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2001, pp. 439–447.
[23] R. Motwani, R. Panigrahy, Y. Xu, Estimating sum by weighted sampling, in: Proceedings of the 34th International Colloquium on Automata, Languages and Programming, 2007, pp. 53–64.
[24] M. Parnas, D. Ron, On approximation the minimum vertex cover in sublinear time and the connection to distributed algorithms, Electronic Colloquium on Computational Complexity (ECCC), 94, 2005.
[25] M. Thorup, Quick k-median, k-center, and facility location for sparse graphs, SIAM Journal on Computing 34 (2) (2004) 405–432.